Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 583926, 9 pages
doi:10.1155/2008/583926

Research Article

Enhanced Montgomery Multiplication on DSP Architectures for Embedded Public-Key Cryptosystems

P. Gastaldo, G. Parodi, and R. Zunino

Department of Biophysical and Electronic Engineering (DIBE), University of Genoa, Via Opera Pia 11a, 16145 Genova, Italy

Correspondence should be addressed to P. Gastaldo, paolo.gastaldo@unige.it

Received 21 September 2007; Revised 16 January 2008; Accepted 27 February 2008

Recommended by Sandro Bartolini

Montgomery's algorithm is a popular technique to speed up modular multiplications in public-key cryptosystems. This paper tackles the efficient support of modular exponentiation on inexpensive circuitry for embedded security services and proposes a variant of the finely integrated product scanning (FIPS) algorithm that is targeted to digital signal processors. The general approach improves on the basic FIPS formulation by removing potential inefficiencies and boosts the exploitation of computing resources. The reformulation of the basic FIPS structure results in a general approach that balances computational efficiency and flexibility. Experimental results on commercial DSP platforms confirm both the method's validity and its effectiveness.

1. INTRODUCTION

Modular exponentiation is the core operation of successful public-key cryptosystems such as RSA [1] and ElGamal encryption [1]; modular arithmetic also plays a key role in elliptic curve cryptography [1]. The RSA scheme requires exponentiation on k-bit positive integers, with k ranging typically from 512 to 2048; the ElGamal scheme uses exponentiation on prime numbers (at least 1024 bits). When dealing with large integers, modular exponentiation typically iterates some modular multiplication algorithm. As a result, fast, space-efficient algorithms for modular multiplication [1, 2] have been developed to achieve efficient implementations.

Montgomery's algorithm [2, 3] can effectively speedup the modular multiplications required during an exponentiation process. The method reformulates the (demanding) algebraic operations brought about by modular multiplication and uses a fixed power of 2 as a computational basis, which best suits digital implementations. Several approaches have been proposed in the literature for the implementation of Montgomery's multiplication [2]. In view of the expanding demand of security services on embedded machinery, great efforts have been recently devoted to developing efficient implementations of that algorithm on FPGAs, DSPs, and microcontrollers [4–15].

The present work focuses on the implementation of Montgomery's algorithm on digital signal processors and proposes an enhanced version of the finely integrated product scanning (FIPS) approach [2]. Among the several alternatives [2], the FIPS strategy best exploits both the multiply-and-accumulate capabilities and the double-path organisation of most DSP platforms.

When comparing this approach with the state-of-the-art on embedded cryptosystems, a few works have addressed recently the reformulation of Montgomery's multiplication for DSPs architectures [9, 12]. The works of Itoh et al. [9] and Krishnamurthy et al. [12] mostly tackled the implementation of a complete public-key cryptosystem and considered one DSP platform exclusively. The implementation aspects of the FIOS and the FIPS algorithm have been discussed in [11, 14], but those papers addressed embedded RISC architectures only.

Thus, to the purpose of improving the efficiency of embedded public-key cryptosystem, the paper reformulates the basic FIPS structure and provides an enhanced algorithm design that matches computational efficiency with flexibility. The approach performs effectively in very long instruction word (VLIW) architectures, and therefore DSPs can be envisioned as the target platform under the (reasonable) assumption that today's DSP devices are built on a VLIW schema.

Inputs: $\{\overline{a}, \overline{b}\}$ (structural fixed parameters: n', r).
Outputs: $\overline{c} = a \cdot b \cdot r \pmod{n}$.
1. $t = \overline{a} \cdot \overline{b}$,
2. $m = t \cdot n' \pmod{r}$,
3. $u = (t + m \cdot n)/r$,
4. if $(u \geq n)$, then return $(u - n)$ else return u.

Algorithm 1: Montgomery's reduction algorithm $\mathrm{MP}(\overline{a}, \overline{b})$.

The paper is organized as follows. Section 2 briefly introduces Montgomery's multiplication and the FIPS algorithm. Section 3 describes the enhanced FIPS formulation and presents the related theoretical analysis. Section 4 presents the results obtained by implementing the reformulated algorithm on different DSP platforms.

2. MONTGOMERY MULTIPLICATION IN EMBEDDED CRYPTOSYSTEMS

2.1. Montgomery's reduction algorithm

For any pair of integers, $a, b < n$, Montgomery's algorithm computes

$$\mathrm{MP}(a, b) = a \cdot b \cdot r^{-1} \bmod n, \tag{1}$$

where r is an integer that is larger than n and is coprime to n, that is, $\mathrm{GCD}(n, r) = 1$, and r^{-1} is the modular inverse of $r : r^{-1} \cdot r = 1 \pmod{n}$. For computational reasons clarified below, $\mathrm{MP}(\cdot)$ is not usually fed with the actual operands, $\{a, b\}$, but processes their so-called n-residual representations, $\{\overline{a}, \overline{b}\}$. The n-residual, $\overline{a}$, of an integer $a < n$ is defined as $\overline{a} = a \cdot r \pmod{n}$. Montgomery's algorithm can be outlined as shown in Algorithm 1.

The procedure is parameterised by the unique number, $n\prime$, that satisfies: $r \cdot r^{-1} - n \cdot n' = 1$; such quantity and the quantity r^{-1} are computed once for each modulus setting by the extended Euclidean algorithm [16]. A critical step in the $\mathrm{MP}(\cdot)$ algorithm involves the division operation at step (3), having r as a denominator. In digital hardware implementations, one chooses r such that $r = 2^k$, hence the ratio operation turns into a bit-shift; such a setting satisfies the constraint $\mathrm{GCD}(n, r) = 1$ because, in practical cryptosystems, n is either a prime number or the product of two odd primes. Thus, k is eventually set such that: $2^{k-1} < n < 2^k$.

Montgomery's algorithm operates in the residual space, since it computes $\mathrm{MP}(\overline{a}, \overline{b})$ and returns the n-residual representation of the product result: $\mathrm{MP}(\overline{a}, \overline{b}) = \overline{c} = a \cdot b \cdot r \pmod{n}$. Therefore, at run-time, the method requires that both operands are mapped into the n-residual domain, and eventually that the result is back-converted to the integer domain; both those phases prove quite demanding from a computational perspective. Thus, Montgomery's algorithm is really effective when those prologue and epilogue procedures are less relevant than the product computation itself. This explains why the $\mathrm{MP}(\overline{a}, \overline{b})$ approach boosts computational performances in modular exponentiation: in that case, many iterated modular multiplications involve a common modulus and do not require repeated back-conversions of results from the residual domain to the original integer representation.

2.2. Digital implementation algorithms

In real implementations of $\mathrm{MP}(\overline{a}, \overline{b})$, very large integers are split and processed on a word-by-word basis. In the following, w and z will denote the word size (in bits) and the number of words required to represent an integer, respectively. Therefore, r takes on the value $r = 2^{zw}$.

The literature classifies implementation approaches according to the sequencing of multiplications and reductions; "integrated" approaches interleave multiplications and reductions, while "product scanning" algorithms compute sequentially the words of the product result. Thus, the finely-integrated product scanning (FIPS) algorithm [2] works out $\mathrm{MP}(\overline{a}, \overline{b})$ by interleaving the computations of products $\overline{a} \cdot \overline{b}$ and $m \cdot n$. Both multiplications follow the product-scanning method, hence the FIPS structure features two nested loops, the outer of which scans the words of the product itself.

Figure 1 outlines the FIPS pseudocode formalized by Koc et al. [2]. Registers S and C are handled by the adder circuitry and hold the sum and the carry values, respectively. The quantity t (as per step (1) of $\mathrm{MP}(\overline{a}, \overline{b})$) accumulates partial results from either $(\overline{a}. \overline{b})$ or $(m \cdot n)$ and is here hosted by an array, $t[\cdot]$, of three registers (each register is of size w bits). Since each inner loop contains multiply-and-accumulate (MAC) operations, the algorithm fits the architectures of DSP devices, which typically provide parallel, fixed-point multipliers.

In the pseudocode, each $\mathrm{ADD}(t[1], C)$ function call at steps O4, O9, O12, and O17 has two crucial effects: (1) the carry C from each summation is added to the array element $t[1]$, and (2) the carry value resulting from the latter addition propagates to the array element $t[2]$. Carry propagation, however, can convey some inefficiency: each of those steps triggers two additions even when the addition $(t[1] + C)$ does not actually generate a carry; in a parallel architecture, this might lead to an imbalanced use of functional units and ultimately affect pipelining. In principle, one might insert *if* statements to detect carry propagation in advance; such a trick, however, would severely compromise computational efficiency, due to pipeline-flush effects generated by conditional branches within the most nested loops. Moreover this approach would expose the system to timing attacks [17].

2.3. Architectural issues in the FIPS algorithm

In the original FIPS formulation, all elements of the array $t[\cdot]$ have equal size, w, favouring implementations on flexible hardware. The method assumes that $z < W$ and, as observed in [2], allocates $\log_w[zW(W-1)]$ words for the accumulator $(W = 2^w)$. DSP-based implementations, however, would more likely support such a structure with one contiguous register of suitable size, K (see Figure 2). Such a simple

```
The basic FIPS algorithm
O1.   for i = 0 to(z − 1)
        beginloop
O2.       for j = 0 to (i − 1)
            beginloop
O3.           (C, S):= t[0] + ā [j]·b̄[i − j]          /*MAC operation, ā·b̄ */
O4.           ADD(t[1], C)
O5.           (C, S):= S + m[j]·n[i − j]              /* MAC operation, m·n */
O6.           t[0]:= S
O7.           ADD(t[1], C)
            endloop
O8.         (C, S) := t[0] + ā [i]·b̄ [0]              /* MAC operation, ā·b̄ */
O9.         ADD(t[1], C)                              /* carry propagation */
O10.        m[i] := S·n′[0]mod w                      /* mod(·) is a truncation */
O11.        (C, S):= S + m[i]·n[0]                    /* MAC operation, m·n */
O12.        ADD(t[1], C)
O13.        t[0]:= t[1]; t[1]:= t[2]; t[2]:= 0
          endloop
O14.      for i = z to (2z − 1)
          beginloop
O15.        for j = (i − z + 1) to (z − 1)
            beginloop
O16.          (C, S):= t[0] + ā [j]·b̄[i, j]          /* MAC operation, ā·b̄ */
O17.          ADD(t[1], C)
O18.          (C, S):= S + m[j]·n [i − j]            /* MAC operation, m·n */
O19.          t[0]:= S
O20.          ADD(t[1], C)
            endloop
O21.        m[i − z]:= t[0]
O22.        t[0]:= t[1]; t[1]:= t[2]; t[2]:= 0
          endloop
```

Figure 1: The basic version of the FIPS algorithm [2].

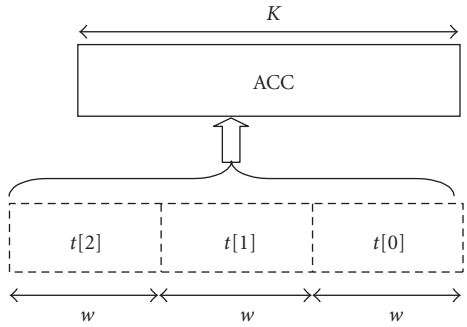

Figure 2: The DSP-oriented version of the FIPS algorithm: single-accumulator register array.

reformulation exploits any extended accumulator and fits two main features of modern DSPs: (1) the lowest section, $t[0]$, of the array holds the w least significant bits and directly implements (mod w) arithmetic; (2) arithmetic circuitry subsumes carry propagation at steps $\{O4, O9, O12, O17\}$, thus avoiding multiple accesses to the array elements. The pseudocode in Figure 3 presents an excerpt of the reformulated FIPS algorithm and shows that one accumulator can boost performance in the innermost j-loop by supporting the MAC operations, $\overline{a}\cdot\overline{b}$ and $m\cdot n$. The single-accumulator configuration requires that the underlying architecture ensures a consistent support of the precision, and the related criterion to design the register size, K, was analysed in [2]. The content of the accumulator ACC can be upper bounded by

$$\mathrm{ACC} \le (2z+2)\left(2^w-1\right)^2 \stackrel{\mathrm{def}}{=} \mathrm{ACC}^{(\mathrm{UP})}. \tag{2}$$

Thus, a design criterion for dimensioning K requires that

$$K > \log_2\left[(2z+2)\left(2^w-1\right)^2\right]. \tag{3}$$

3. ENHANCED DSP-ORIENTED DESIGN OF THE FIPS ALGORITHM

In spite of the apparent trivial reformulation, the constraint (3) may give rise to some architectural issues when embedding the FIPS algorithm on commercial DSPs. In such cases, the manufacturer's choices set the type and the number of available multipliers; performance optimisation now clearly requires that the quantity, w, matches the word size of the multipliers available on the target DSP platform. As a result,

```
Intermediate FIPS algorithm
E1.   for i = 0 to z − 1
        beginloop
E2.       for j = 0 to i − 1
            beginloop
E3.           ACC = ACC + ā[j]·b̄[i − j]
E4.           ACC = ACC + m[j]·n[i − j]
            endloop
E5.         ACC = ACC + ā[i]·b̄[0]
E6.         m[i] = ACC·n′[0] mod w
E7.         ACC = ACC + m[i]·n[0]
E8.         ACC = ACC ≫ w
        endloop
E9.   for i = z to 2z − 1
        (· · · )
```

Figure 3: FIPS algorithm: code for the single-accumulator configuration.

Table 1: Required width of the extended accumulator in the formulation of the FIPS algorithm.

Multiplier circuitry-word size	Cryptosystem configuration		
	512 bit	1024 bit	2048 bit
16 bit	39	40	41
32 bit	69	70	71

multiplier specifications eventually set the register size, K, for the single-accumulator configuration.

Table 1 gives the required width, K, of the extended accumulator when using 16-bit and 32-bit multipliers for three typical public-key settings (512, 1024, and 2048 bits). For example, a 2048-bit cryptosystem requires an accumulator holding at least 71 bits on a 32-bit DSP. The table actually points out that the single-accumulator reformulation implies hardware specifications that can hardly fit commercial fixed-point DSPs.

The research presented in this paper improves on the basic FIPS algorithm by removing these potential sources of inefficiency under two constraints: to exploit only registers of size w and $2w$ and to keep computational efficiency unaffected as compared with the single-accumulator configuration. The proposed structure allocates four registers, each having size $2w$: registers *REG*1 and *REG*2 accumulate results from MAC operations $\overline{a}\cdot\overline{b}$ and $m\cdot n$, respectively, in the inner loop; ACCX stores the associate carry values; register *ACCY* finally accumulates the partial results stored in REG1 and REG2 and prepares the computation of $m[i]$ in the outer loop at step O10. This configuration matches the double-path organisation of DSP platforms by decoupling the two MAC operations in the inner loop.

A more significant enhancement to the basic FIPS algorithm reformulates the structure of outer loop. The original code section shows that the computation of the value $m[i]$ completes at line O10 and is subsequently used at line O11. Such a dependency might clearly affect efficiency by preventing a full exploitation of the processor functional units. Two adjustments to the critical code section overcome that dependency: first, the dependency is removed by pipelining the inter-dependent operations across two consecutive loop cycles; secondly, the multiple-accumulator structure (REG1 and REG2 in the inner loop, ACCY in the outer loop) favours parallelism in the computation of intermediate results. Such an optimisation applies only to the first i-loop of the FIPS algorithm, as the second i-loop does not suffer from that dependency.

The pseudocode in Figure 4 outlines the complete version of the enhanced FIPS algorithm for parallel architectures. The outer loop (EE4-EE20) starts by storing in ACCY the higher part of the eventual result for the MAC operation (EE5), which uses the contributions (Q and m) from the previous (i–1)th iteration; such a configuration actually eliminates the dependency between line O10 and O11 of the original algorithm. Then, register ACCY accumulates partial results from a MAC operation (EE7) and the inner loop (REG1 and REG2); the summation at line EE15 prepares the completion of $m[i]$ at line EE17.

As a result of the optimization, the inner loop EE9–EE14 preserves a MAC-parallel structure, while the overall structure can be supported without a single, three-word long accumulator. The reformulation clearly requires the introduction of a prologue section at lines EE1–EE3.

4. EXPERIMENTAL RESULTS

The efficiency performance of the enhanced FIPS implementation was evaluated experimentally on commercial fixed-point DSP devices. To assess the general validity of the enhancements, the practical tests involved two different families of platforms, namely, the Texas Instruments TMS320C6201 device [18] and the Analog Devices ADSP-TS201S "TigerSHARC" processor [19]. Both DSPs feature very long instruction word (VLIW) architectures. The former DSP [18] offers two independent data paths and supports up to eight instructions in parallel; the eight independent functional units include a pair of 16-bit multipliers and six ALUs (32/40 bit). The load-store architecture is supported by thirty-two 32-bit general-purpose registers, and the common off-register memory hierarchy consists of a 32-bit address space partially mapped into an on-chip RAM. The latter device, TigerSHARC [19], features a static superscalar architecture supporting a computation pipeline, dual computation blocks, and can execute up to four instructions per cycle. Each computation block hosts an ALU, a multiplier, a shifter, a 32-word register file, and a communications logic unit (CLU); multipliers can operate both on 16-bit and 32-bit operands, and accumulators support 64-bit integers.

To attain a reliable comparison of performances, both the enhanced formulation and the single-accumulator configuration of the FIPS algorithm were tested on both DSP platforms. The experimental results for each platform will be reported separately in the following.

The enhanced FIPS algorithm for parallel architectures

EE1. REG1:= $\overline{a}[0]\cdot\overline{b}[0]$
EE2. Q:= REG1 mod w
EE3. REG1:= REG1 $\gg w$
EE4. for $i = 1$ to $z - 1$
beginloop
EE5. ACCY:= $Q + m[i-1]\cdot n[0]$
EE6. ACCY:= ACC$Y \gg w$
EE7. ACCY:= ACCY + $\overline{a}[i]\cdot\overline{b}[0]$
EE8. for $j = 0$ to $i - 1$
beginloop
EE9. REG1:= REG1 + $\overline{a}[j]\cdot\overline{b}[i-j]$
EE10. ACCX:= ACCX + (REG1 $\gg w$)
EE11. REG1:= REG1 mod w
EE12. REG2:= REG2 + $m[j]\cdot n[i-j]$
EE13. ACCX:= ACCX + (REG2 $\gg w$)
EE14. REG2:= REG2 mod w
endloop
EE15. ACCY:= ACCY + REG1 + REG2
EE16. Q:= ACCY mod w
EE17. $m[i]$:= $Q\cdot n'[0]$ mod w
EE18. REG1:= ACCX mod w
EE19. ACCX:= ACC$X \gg w$
EE20. REG2:= ACC$Y \gg w$
endloop
EE21. ACCY:= $Q + m[i-j]\cdot n[0]$
EE22. REG2:= REG2 + (ACCY$\gg w$)
EE23. for $i = z$ to $2z - 1$
beginloop
EE24. for $j = (i - z + 1)$ to $(z - 1)$
beginloop
EE25. REG1:= REG1 + $\overline{a}[j]\cdot\overline{b}[i-j]$
EE26. ACCX:= ACCX + (REG1 $\gg w$)
EE27. REG1:= REG1 mod w
EE28. REG2:= REG2 + $m[j]\cdot n[i-j]$
EE29. ACCX:= ACCX + (REG2 $\gg w$)
EE30. REG2:= REG2 mod w
endloop
EE31. REG2:= REG2 + REG1
EE32. $m[i-z]$:= REG2 mod w
EE33. REG1:= ACCX mod w
EE34. ACCX:= ACC$X \gg w$
EE35. REG2:= REG2 $\gg w$
endloop

Figure 4: The enhanced FIPS algorithm.

4.1. TMS320C6201 DSP platform

The TMS320C6201 device [18] provides 16-bit multipliers and supports up to 40-bit data size. As a result of the design issues discussed in the previous section, those hardware specifications lead to using a word size of at most $w = 16$ bits for FIPS-based cryptosystems. Table 1 shows that, due to the bound (3) on the accumulator size, the DSP device could support the single-accumulator configuration for at most a 1024-bit cryptosystem having $z = 64$ words. Thus, the performance of the enhanced formulation of FIPS has been compared with that of the basic FIPS for a 1024-bit configuration.

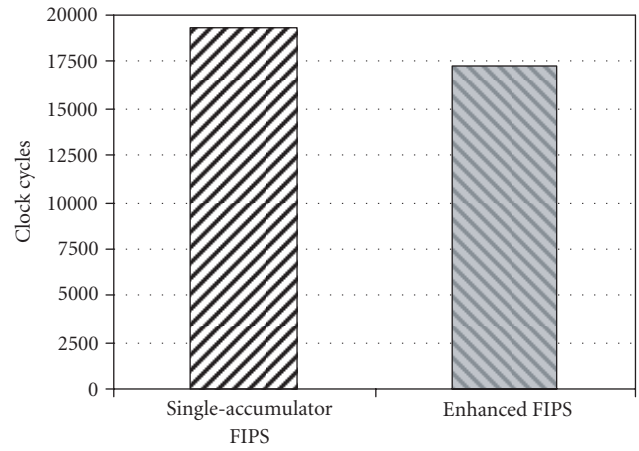

Figure 5: Performance comparison between the two FIPS versions on TMS320C6201 for a 1024-bit implementation of modular multiplication.

The computational cost to work out MP$(\overline{a}, \overline{b})$ allowed one to assess the overall increase in efficiency provided by the algorithm optimisation. For the TMS320C6201 device, the original FIPS algorithm exploiting the single-accumulator configuration required 19372 clock cycles, whereas the enhanced reformulation completed in 17346 clock cycles, thus yielding a 11% reduction in running time; Figure 5 reports on these results. This proved that the enhanced FIPS algorithm improved on the implementation performance without even exploiting the single-accumulator configuration. Such a result confirmed that the reformulation of the FIPS algorithm successfully combined flexibility and efficiency.

A functional explanation of these improvements in performance can be obtained by using the resource-utilisation feedback reports [20], provided by the TMS320C6201 compiler for the two FIPS implementations. Five quality parameters characterize the results of loop pipelining [20]:

(i) *loop carried dependency bound* ≡the minimum interval due to dependencies among quantities across different loop iterations (a dependency occurs when one iteration of a loop writes a value that must be read in a future iteration),
(ii) *unpartitioned resource bound* ≡the smallest iteration delay before the dispatching of instructions to either data path,
(iii) *partitioned resource bound* ≡the smallest iteration delay after the dispatching of instructions to either data path,
(iv) *iteration interval* ≡the number of clock cycles between the start of consecutive loop iterations,
(v) *iterations in parallel* ≡the pipeline depth.

Table 2 compares the compiler feedback reports relative to the inner loops of each FIPS implementations. As expected, the original FIPS algorithm in the single-accumulator configuration shows performs slightly better in terms of inner-loop pipelining; this is due to the fact that

Table 2: Feedback report for the TMS320C6201.

Code quality parameter	Single ACC. FIPS	Enhanced FIPS
Loop carried dependency bound	0	2
Unpartitioned resource bound	2	3
Partitioned resource bound	2	3
Iteration interval	2	3
Iterations in parallel	5	4

the partial results of MAC operations accumulate in a single register. This confirms that the inner-loop design undergoes a tradeoff between flexibility and efficiency.

The measures reported in Table 3 confirm that the inner loop design of the FIPS reformulation can profitably balance the resource management, although it obviously increases the number of instructions dispatched to some specific units.

Overall, compiler-feedback reports prove that the enhanced version of FIPS improves on the global computational performance of the single-accumulator configuration, even when the latter can take advantage of a most efficient design of the inner loop.

The effectiveness of the proposed reformulation of FIPS is indeed confirmed by the results obtained from the experimental session involving a 2048-bit cryptosystem. In this case, as per Table 1, the hardware specifications of the DSP device do not allow the implementation of the single-accumulator configuration, since $w = 16$ bits and $z = 128$ words. Thus, Figure 6 reports on the computational cost to work out MP$(\overline{a}, \overline{b})$ for a standard FIPS implementation and for the enhanced algorithm proposed by this research. The standard FIPS algorithm required 113610 clock cycles to compute MP$(\overline{a}, \overline{b})$, whereas the enhanced reformulation completed in 63294 clock cycles, thus yielding a 45% reduction in running time.

4.2. ADSP-TS201S DSP platform

The "TigerSHARC" DSP [19] supports fixed-point multiply and accumulates operations with 16-bit or 32-bit input operands and a configurable result accumulator of 32, 40, or 64 bits. These hardware specifications virtually remove any practical limit to the word length of the supported cryptosystems when a word size $w = 16$ bits is adopted. On the other hand, target implementations of FIPS having word size $w = 32$ bits cannot rely on the single-accumulator configuration, for example, Table 1 shows that a 64-bit accumulator could not even support a 512-bit cryptosystem.

Therefore, to compare performances consistently, the experiments targeted a 1024-bit cryptosystem and involved three different FIPS implementations:

(i) a single-accumulator configuration, with $w = 16$ bits (thus, $z = 64$ words),
(ii) a basic FIPS algorithm, with $w = 32$ bits (thus, $z = 32$ words),
(iii) the enhanced reformulation, with $w = 32$ bits (thus, $z = 32$ words).

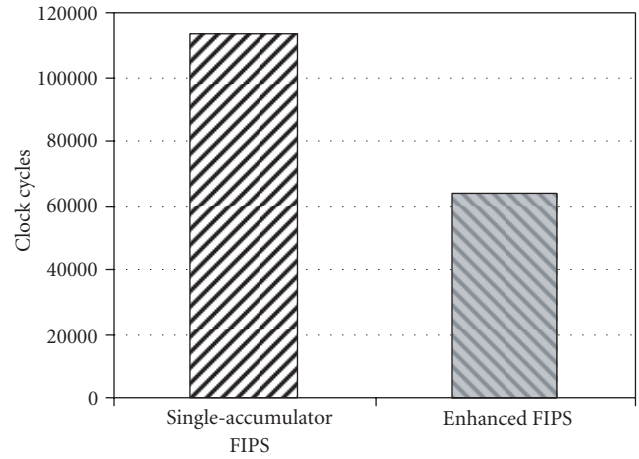

Figure 6: Performance comparison between the two FIPS versions on TMS320C6201 for a 2048-bit implementation of modular multiplication.

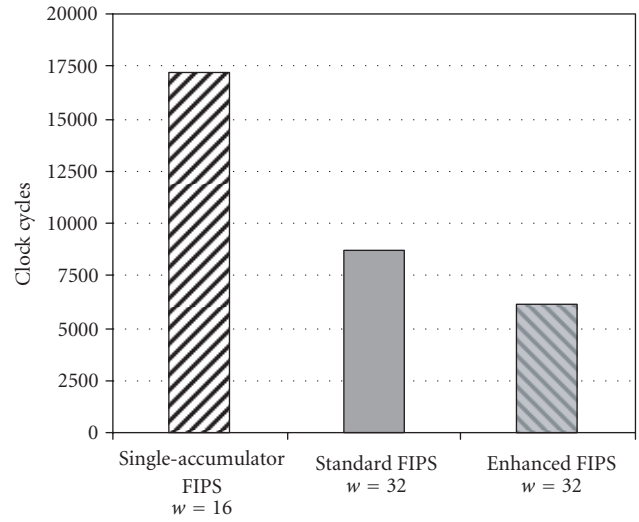

Figure 7: Performance comparison between the three FIPS versions on TigerSHARC for a 1024-bit implementation of modular multiplication.

Figure 7 compares the computational costs to work out MP$(\overline{a}, \overline{b})$ for the three cases. The single-accumulator configuration required 17246 clock cycles; the standard 32-bit FIPS required 8694 clock cycles, whereas the enhanced reformulation completed in 6131 clock cycles, thus yielding a 65% and a 30% reduction in running time, respectively. These results indeed confirm the effectiveness of the enhanced design of FIPS.

The empirical performances of the Analog Device platform were interpreted by using the feedback report from the TS201S compiler [21], including quality parameters obtained from *modulo scheduling*. Table 4 reports on the measured results, related to the inner loops of the two FIPS implementations using $w = 32$ bits, namely, lines O3–O7 and EE9–EE14 for the original version and the enhanced

Table 3: Resource usage report for the TMS320C6201.

	Single ACC. FIPS		Enhanced FIPS	
	A	B	A	B
.L units compares/long data arithmetic	1	1	0	0
.S units shift/branch/ALU/field op.	1	0	2	1
.D units data/addition/subtraction	2	2	2	2
.M units multiply	1	1	1	1
.X cross paths	1	1	2	2
Logic ops (.LS)	0	0	1	1
Addition ops (.LSD)	0	1	3	2
Bound (.L .S .LS)	1	1	2	1
Bound (.L .S .D .LS .LSD)	2	2	3	2

Table 4: Feedback report from the ADSP-TS201S compiler.

Code quality parameter	Original FIPS	Enhanced FIPS
In Int	6	4
SC	3	4
MVE unroll	1	1
res MII	0	0
rec MII	6	4
Cycle count (stalls)	7 (1)	3 (0)

version, respectively. Quality performance is described by the following parameters [21]:

(i) *initiation interval* (InInt) ≡ the number of cycles between the starting of two consecutive loop iterations;
(ii) *stage count* (SC) ≡ the number of initiation intervals until the first iteration of the loop has completed;
(iii) *modulo variable expansion unroll factor* (MVE unroll) ≡ the number of times the loop must be unrolled so that the scheduling has no overlapping register lifetimes;
(iv) *minimum initiation interval due to resources* (res MII) ≡ the lower limit for the initiation interval, imposed by the fact that at least one of the resources is used at maximum capacity;
(v) *minimum initiation interval constrained by recurrences* (rec MII) ≡ the lower limit for the initiation interval imposed by the recurrences in the code;
(vi) *cycle count* ≡ the number of clock cycles required to execute one iteration of the loop, including stalls; the number of stalls is given between brackets.

Empirical evidence witnesses the efficiency of the enhanced implementation, as the latter version outperformed the basic implementation. Those results explain the significant increase in computational speed measured experimentally.

Table 5 reports on the experimental measures in terms of resource utilisation; for each resource, the table gives the percentage of utilisation during the entire loop. The *sequencer* parameter is associated with the programme sequencer, entrusted with address dispatching and efficient scheduling of arithmetic operations. *Instruction slots* count the accesses to the slots providing the instructions. *JALU* and *IALU* refer to the integer ALU registers, which in general store operands and intermediate or final results of integer computations. Finally, *compute block*, *compute block ALU*, *compute block multiplier,* and *compute block shifter* measure the resource usage of the two independent processing blocks (X and Y) supported by the TS201S DSP.

Table 5: Resource utilisation report for the ADSP-TS201S.

	Original FIPS	Enhanced FIPS
Sequencer	14.3%	25.0%
Instruction slots	67.9%	100.0%
IALU	71.4%	62.5%
JALU	57.1%	100.0%
X compute block	—	87.5%
X compute block ALU	—	100.0%
X compute block multiplier	—	25.0%
X compute block shifter	28.6%	50.0%
Y compute block	57.1%	37.5%
Y compute block ALU	57.1%	50.0%
Y compute block multiplier	28.6%	25.0%
Ureg move/Immediate load	42.9%	12.5%

The feedback from the compiler shows that the proposed implementation of FIPS attained a more efficient management of the compute blocks X and Y. The arithmetic scheduling exhibited a much better balance, whereas the original FIPS version tended to stress unit Y to the disadvantage of its counterpart X which mostly remained idle.

The overall efficiency of the proposed implementation of FIPS is validated by the results obtained with the experiment involving a 2048-bit cryptosystem. Figure 8 compares the computational costs to work out $\mathrm{MP}(\overline{a}, \overline{b})$ for the three cases. The single-accumulator configuration ($w = 16$ bits, $z = 128$ words) required 59104 clock cycles; the standard 32-bit FIPS

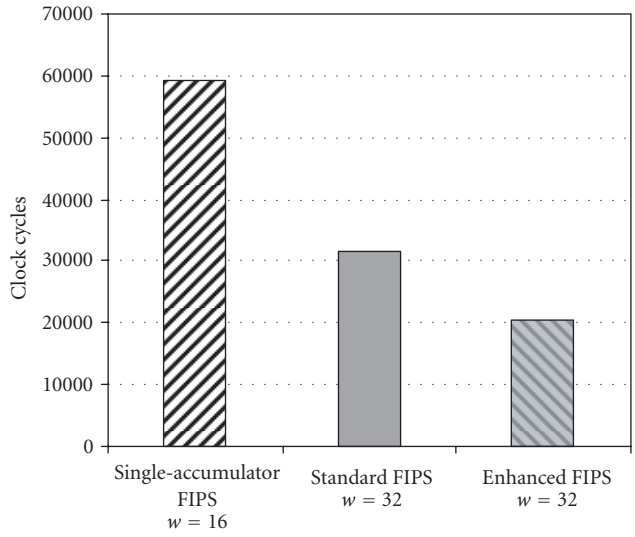

Figure 8: Performance comparison between the three FIPS versions on TigerSHARC for a 2048-bit implementation of modular multiplication.

($w = 32$ bits, $z = 64$ words) required 31575 clock cycles, whereas the enhanced reformulation completed in 20348 clock cycles, thus yielding a 65% and a 35% reduction in running time, respectively.

5. CONCLUSIONS

The reformulation of modular reduction in Montgomery's algorithm simplifies circuitry via a radix-2 representation but is not necessarily powerful enough per se to boost efficiency and requires some careful attention when implementing the basic theoretical principle. The various versions of Montgomery's algorithm [2] witness indeed the intricacies conveyed by the possible strategies in scheduling modular operations.

The method proposed in this paper was aimed at supporting Montgomery's product on DSP platforms by means of the FIPS algorithm and exploits the multiply-and-accumulate capabilities and the double-path organisation typical of most DSP devices. Hence, the paper reformulated the basic FIPS version of Montgomery's product for matching computational efficiency with flexibility, without affecting realisation costs. A general design criterion confirmed that the reformulated approach fits today's commercial devices for embedded cryptosystems.

Experimental results verified the performances of the proposed approach on two DSP families characterized by different hardware specifications. The method scored a notable performance in terms of computational speed and resource balancing; this proved overall effectiveness and the general validity of the approach, considering that those results were obtained on quite different platforms.

Although the proposed method seems to best fit DSP architectures, the reformulation of the FIPS algorithm still applies effectively to FPGA-supported setups. In the latter cases, other versions of Montgomery's product seem to perform better [22]; those approaches exhibit a peculiar structure of the internal loops, which may not benefit from the enhancements to the FIPS formulation described in this paper.

REFERENCES

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Fla, USA, 1997.

[2] C. K. Koc, T. Acar, and B. S. Kaliski Jr., "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, 1996.

[3] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[4] S. R. Dussè and B. S. Kaliski Jr., "A cryptographic library for the Motorola DSP56000," in *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology (EUROCRYPT '90)*, pp. 230–244, Aarhus, Denmark, May 1990.

[5] S. E. Eldridge and C. D. Walter, "Hardware implementation of Montgomery's modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 42, no. 6, pp. 693–699, 1993.

[6] C. K. Koc, "RSA hardware implementation," Tech. Rep. 801, RSA Laboratories, Redwood City, Calif, USA, 1996, http://islab.oregonstate.edu/koc/papers/reports.html.

[7] A. Royo, J. Moran, and J. C. Lopez, "Design and implementation of a coprocessor for cryptography applications," in *Proceedings of the European Design & Test Conference (EDTC '97)*, pp. 213–217, Paris, France, March 1997.

[8] C.-C. Yang, T.-S. Chang, and C.-W. Jen, "A new RSA cryptosystem hardware design based on montgomery's algorithm," *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 7, pp. 908–913, 1998.

[9] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast implementation of public-key cryptography on a DSP TMS320C6201," in *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES '99)*, pp. 61–72, Worcester, Mass, USA, August 1999.

[10] A. F. Tenca and C. K. Koc, "A scalable architecture for modular multiplication based on Montgomery's algorithm," *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1215–1221, 2003.

[11] J. Großschädl and G.-A. Kamendje, "Architectural enhancements for montgomery multiplication on embedded RISC processors," in *Proceedings of the 1st International Conference on Applied Cryptography and Network Security (ACNS '03)*, vol. 2846 of *Lecture Notes in Computer Science*, pp. 418–434, Kunming, China, October 2003.

[12] A. Krishnamurthy, Y. Tang, C. Xu, and Y. Wang, "An efficient implementation of multi-prime RSA on DSP processor," in *Proceedings of the International Conference on Accoustics, Speech, and Signal Processing (ICASSP '03)*, vol. 2, pp. 413–416, Hong Kong, April 2003.

[13] A. Z. Alkar and R. Sönmez, "A hardware version of the RSA using the Montgomery's algorithm with systolic arrays," *Integration, the VLSI Journal*, vol. 38, no. 2, pp. 299–307, 2004.

[14] J. Großschädl, K. C. Posch, and S. Tillich, "Architectural enhancements to support digital signal processing and public-key cryptography," in *Proceedings of the 2nd Workshop on Intelligent Solutions in Embedded Systems (WISES '04)*, pp. 129–143, Graz, Austria, June 2004.

[15] L. Hars, "Applications of fast truncated multiplication in cryptography," *EURASIP Journal of Embedded Systems*, vol. 2007, Article ID 61721, 9 pages, 2007.

[16] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, Mass, USA, 2nd edition, 1981.

[17] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '96)*, vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113, Springer, Santa Barbara, Calif, USA, August 1996.

[18] Texas Instruments, *TMS320C6000 Technical Brief*, SPRU197d, http://focus.ti.com/lit/ug/spru197d/spru197d.pdf.

[19] Analog Devices, *TigerSHARC Embedded Processors*, http://www.analog.com/UploadedFiles/Data_Sheets/ADTS201S.pdf.

[20] Texas Instruments, *TMS320C6000 Optimizing Compiler User's Guide*, SPRU187n, //focus.ti.com/lit/ug/spru187n/spru187n.pdf.

[21] Analog Devices, *C/C++ Compiler and Library Manual for TigerSHARC Processors*, http://www.analog.com/UploadedFiles/Associated_Docs/193758530Compiler_MN_TSxxx_rev_4_0_date_08_07.pdf".

[22] C. McIvor, M. McLoone, and J. V. McCanny, "FPGA montgomery multiplier architectures—a comparison," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*, pp. 279–282, Napa, Calif, USA, April 2004.